

How to create a simple ColdFire and Altera FPGA IOC

W. Eric Norum

March 10, 2006

Contents

1	Introduction	1
2	FPGA application	3
3	EPICS application	9
4	Remote Reset	13
5	Remote Reprogramming	15
5.1	uCDIMM ColdFire 5282 external flash	15
5.2	Development kit configuration flash	16
A	ledDeviceSupport.c	23
B	Alternate Pinouts	27

Chapter 1

Introduction

This tutorial presents a step-by-step series of operations for creating a simple EPICS application for an Arcturus uCDIMM ColdFire 5282 module attached to an Altera FPGA development kit. The following software and hardware components are assumed to be in place:

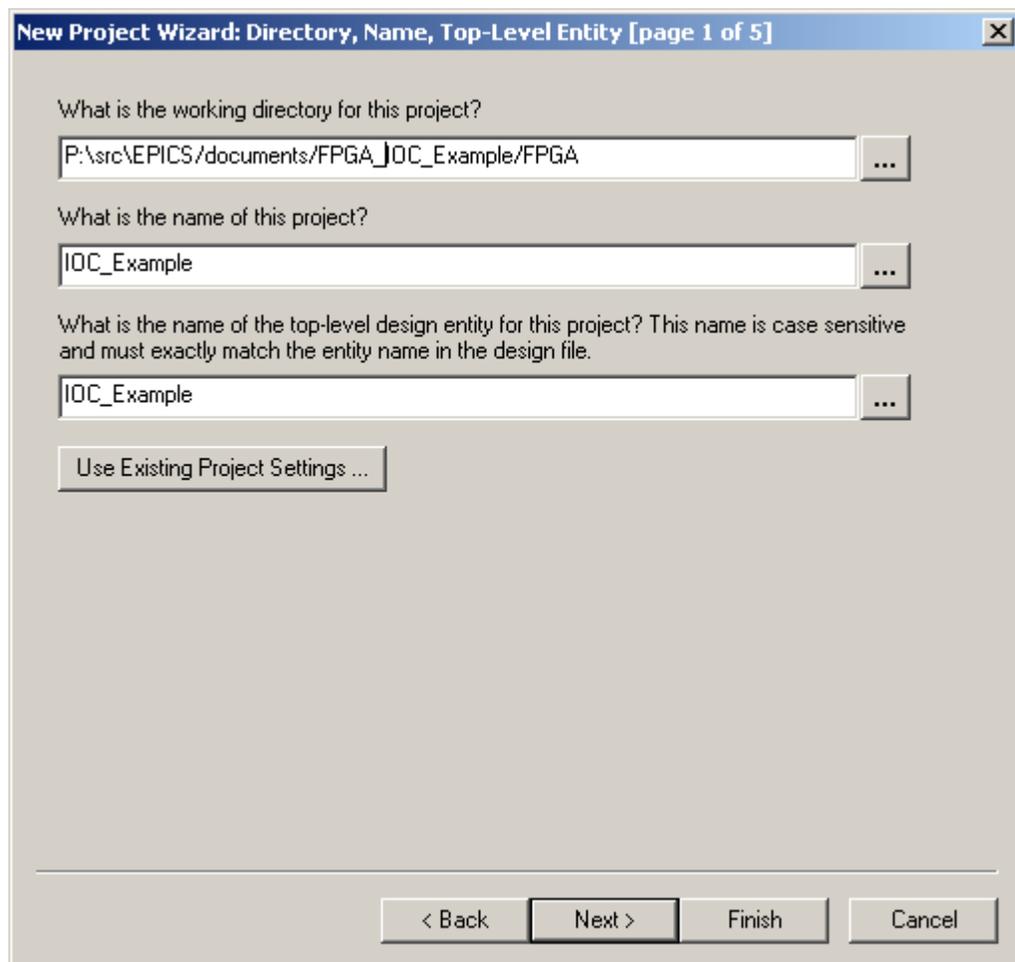
- Arcturus uCDIMM ColdFire 5282
- RTEMS with m68k tool chain and uC5282 board-support package
- EPICS version R3.14.7 or greater with RTEMS-uC5282 target
- Altera FPGA development kit with at least two sets of expansion prototype connectors.
- ColdFire/Stratix adapter card
- Quartus version 5.0 or greater and Altera SOPC builder
- ColdFire bridge SOPC component
- Console reset detect Quartus component (optional)

This tutorial is written for use with an Altera Stratix II DSP development kit. The changes required for use with other development kits are summarized in appendix B.

Chapter 2

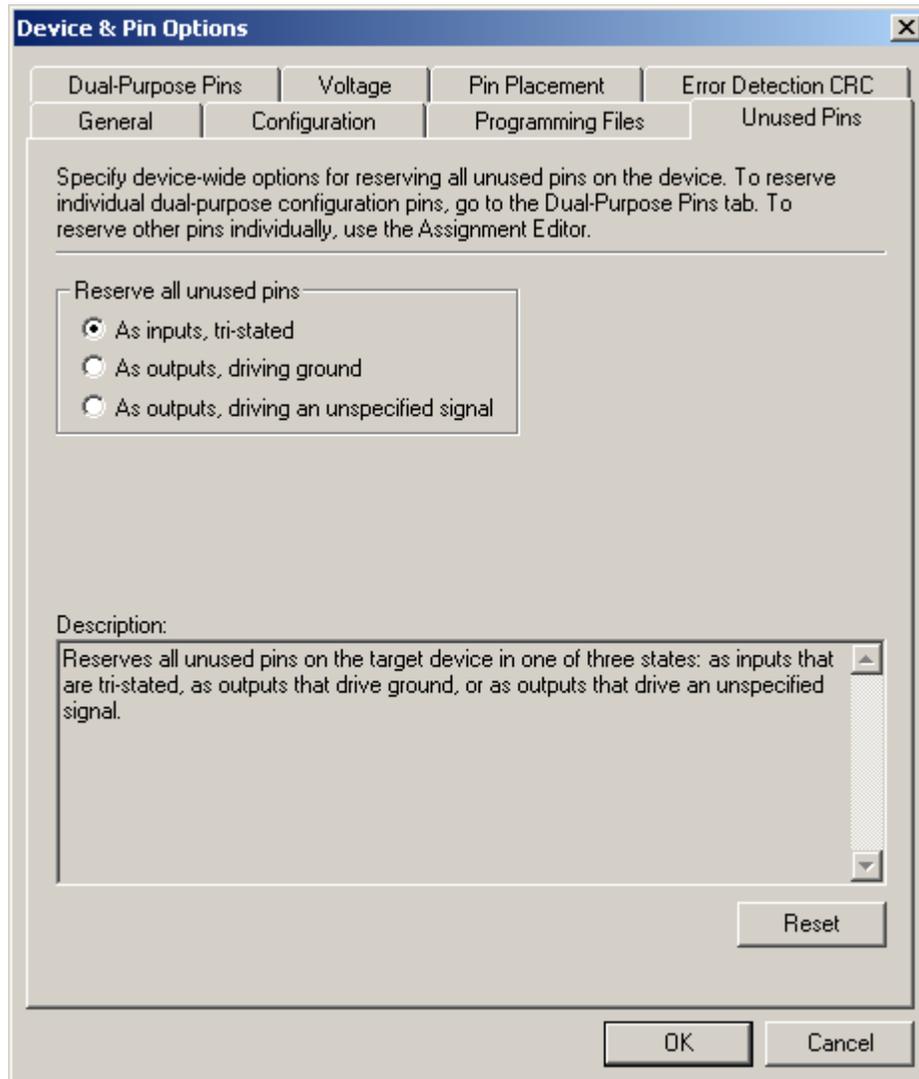
FPGA application

1. Use the Quartus "New Project Wizard" to create a new project. I called the project IOC_Example with same name for the top-level entity:



2. Ensure that unused pins are treated as inputs (Assignments→Device..., Device & Pin Options, Unused Pins tab). Not all the ColdFire signals are used by this example and Quartus helpfully grounds all the corresponding

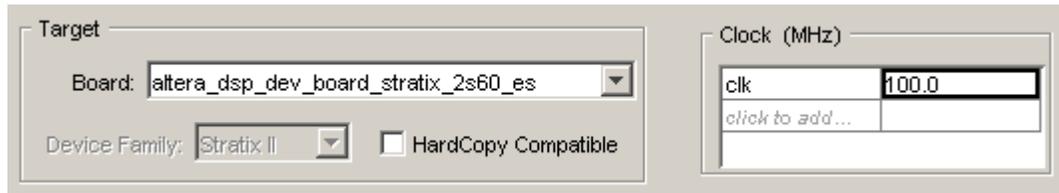
pins if this step is omitted!



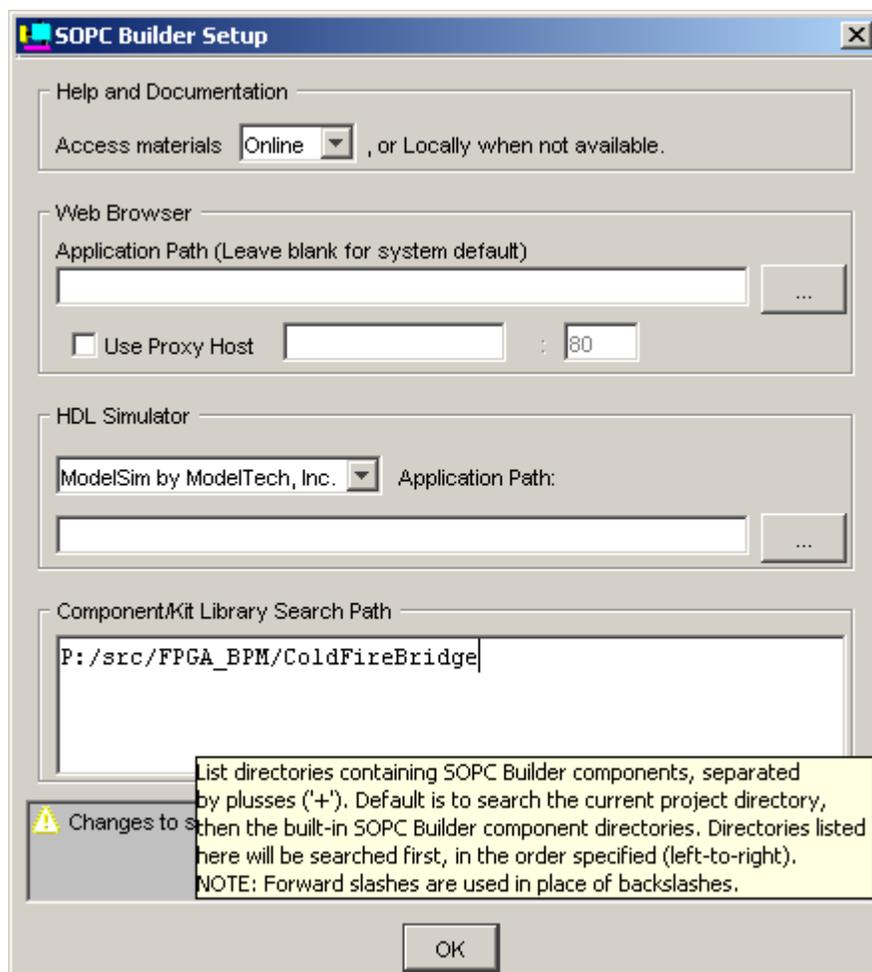
3. Start the SOPC Builder (Tools→SOPC Builder...) and create a new SOPC system. I called the SOPC system IOC_ExampleSystem:



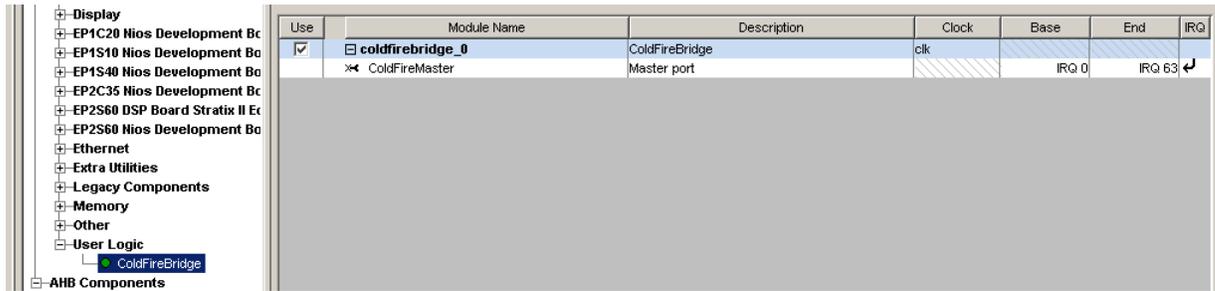
4. Set the SOPC Board and Clock parameters:



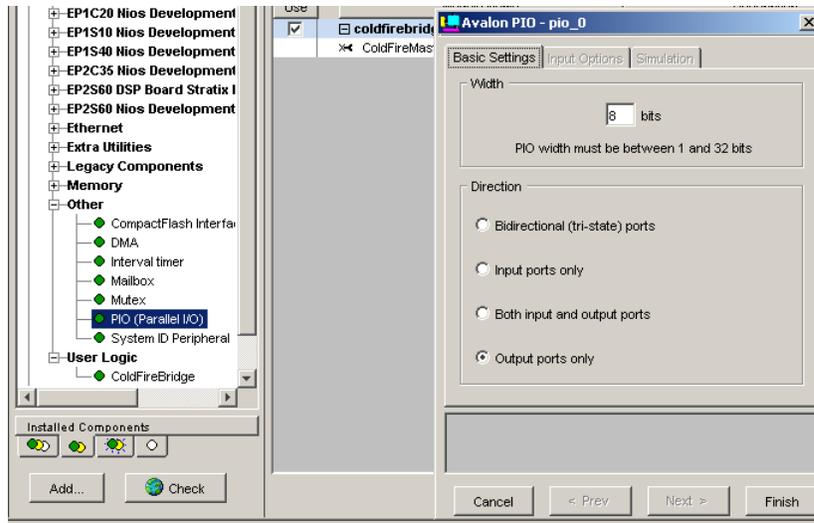
5. If you have not done so in a previous SOPC project, add the directory containing the ColdFire Bridge SOPC component to the list of directories which the SOPC Builder will search (File→SOPC Builder Setup. . .). Note that this list of directories applies to SOPC Builder sessions in **all** projects. Note also that changes to this list do not take effect until the SOPC Builder is next started. When the SOPC Builder is started and searches the directory containing the ColdFire Bridge SOPC component it will add the ColdFire Bridge component to the list of User Logic components.



6. Double-click the ColdFire Bridge component to add it to the SOPC design:



7. Add an 8-bit output port to the SOPC design:



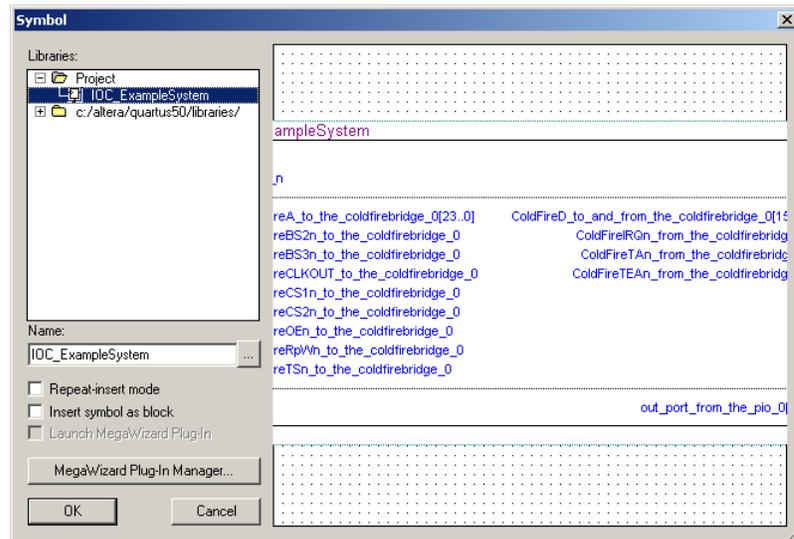
The SOPC components and connections should then be:

Use	Module Name	Description	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>	coldfirebridge_0	ColdFireBridge	clk			
	ColdFireMaster	Master port		IRQ 0		IRQ 63
<input checked="" type="checkbox"/>	pio_0	PIO (Parallel I/O)	clk	0x0000		0x0007

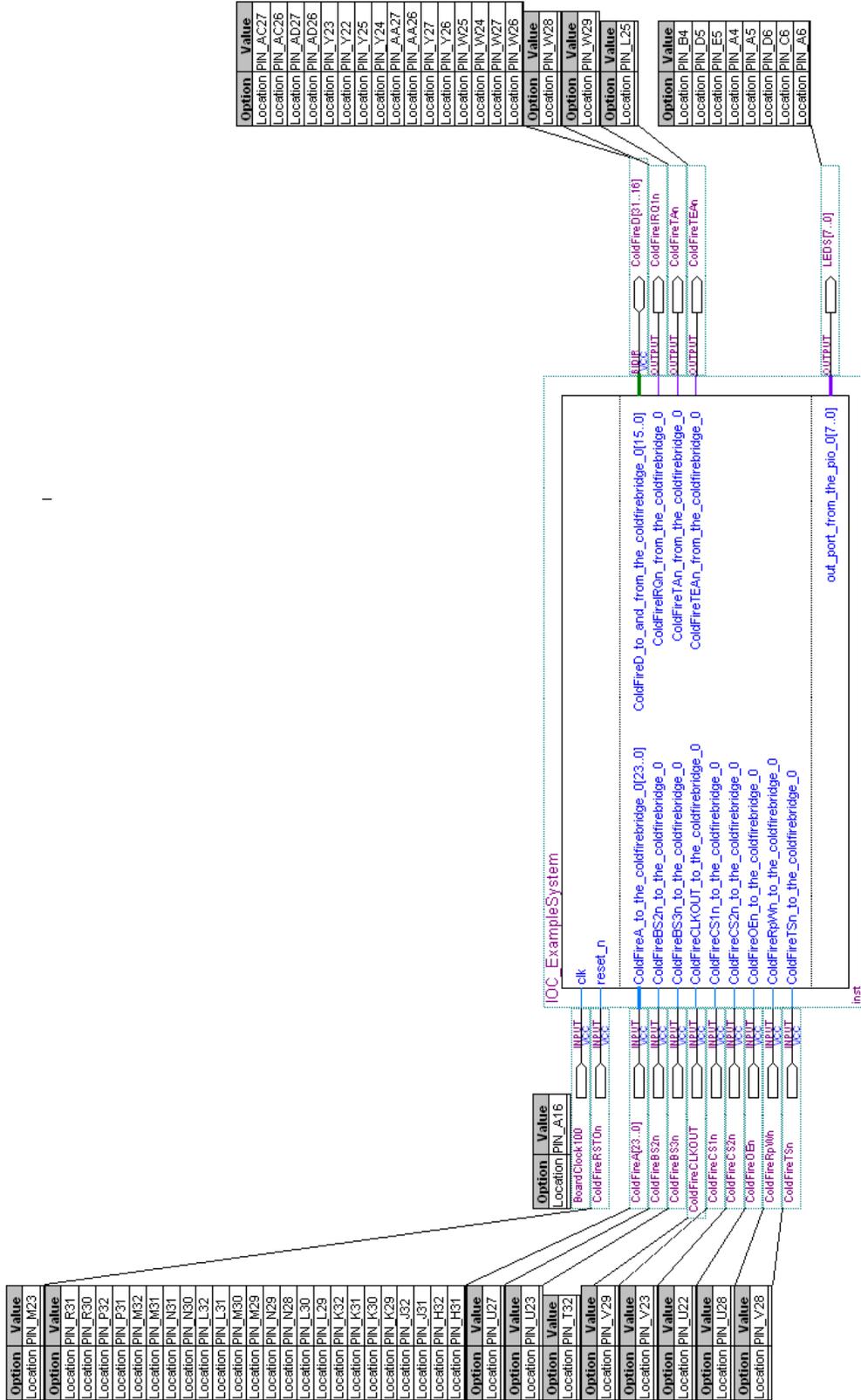
8. Click the **Generate** button to create the SOPC block.

9. Create the top-level entity design file. In the Quartus window select **File**→**New**, create a new **Block Diagram/Schematic File** and save it as **IOC_Example**.

10. Double-click in the **IOC_Example.bdf** design window and add the **IOC_ExampleSystem** SOPC block to the design:



11. Add I/O pins and assign them to the appropriate FPGA pins using the Assignment Editor. A list of the pin assignments for several different development kits is included in appendix B. The complete system should then appear as shown on the following page.
12. Compile the project (you'll see lots of warnings...) and load it into the FPGA.



Chapter 3

EPICS application

The steps listed below show how to create an example EPICS IOC application which uses the ASYN I/O environment to control the LEDs on the FPGA card.

1. Create a new EPICS <TOP> directory and make a new application in it. You must specify the full path to the makeBaseApp.pl script in your EPICS installation:

```
./.../makeBaseApp.pl -t ioc ledDriver  
./.../makeBaseApp.pl -t ioc -i -a RTEMS-uC5282 ledDriver
```

2. Edit configure/CONFIG to enable only the RTEMS-uC5282 target:

```
CROSS_COMPILER_TARGET_ARCHS = RTEMS-uC5282
```

3. Edit configure/RELEASE to specify the location of ASYN support:

```
ASYN=./.../modules/soft/asyn
```

4. Edit ledDriverApp/src/Makefile:

- Build only for RTEMS IOC targets (change the PROD_IOC line to PROD_RTEMS):

```
PROD_RTEMS = ledDriver
```

- Add asyn support:

```
ledDriver_DBD += asyn.dbd
```

- Add the asyn library to ledDriver_LIBS (before the EPICS_BASE_IOC_LIBS line which is already there):

```
ledDriver_LIBS += asyn  
ledDriver_LIBS += $(EPICS_BASE_IOC_LIBS)
```

- Add the FPGA device support dbd file (to be written in a following step):

```
ledDriver_DBD += ledDeviceSupport.dbd
```

- Add the FPGA device support source file (to be written in a following step):

```
ledDriver_SRCS += ledDeviceSupport.c
```

5. Edit ledDriverApp/Db/Makefile and add the line:

```
DB += ledDriver.db
```

6. Create the `ledDriverApp/Db/ledDriver.db` file referred to in the previous step. The file should contain:

```
record(longout, "leds") {
    field(DTYP, "asynInt32")
    field(OUT, "@asyn(ledDriver 0 0)")
}
```

The three values in the `OUT` field are:

- (a) The port name.
- (b) The address (unused by this driver, but still needed)
- (c) The timeout value, in seconds (unused by this driver, but still needed).

7. Create `ledDriverApp/src/ledDeviceSupport.dbd` with the contents:

```
registrar(ledDriverDeviceSupportRegistrar)
```

8. Create `ledDriverApp/src/ledDeviceSupport.c`. A complete listing of is included in appendix A. Much of this file is common to all ASYN drivers. The following points describe the lines of particular interest to this application.

- 12** When the SOPC Builder generates a system it creates a C header file describing the system components. This header file can be included explicitly or, as I have done, can be symbolically linked to a file in the application source directory:

```
ln -s ../../../../FPGA/coldfirebridge_0_map/system.h ledDriverApp/src/SOPC.h
```

- 14** The SOPC address space appears in the ColdFire address space at the locations mapped to $\overline{CS1}$ and $\overline{CS2}$. The RTEMS board-support package sets up these chip selects at locations 30000000_{16} and 31000000_{16} , respectively.
- 15** The '+1' is required because the ColdFire bus is 16-bit big-endian so the least-significant byte on the data bus appears at an odd address.
- 23** The table of `asynInt32` methods can't be statically initialized (like the `asynCommon` methods are at line 55) since the `asynInt32Base` initialize method (invoked at line 99) writes to the table to override some methods.
- 41** The check for the existence of the I/O port turns off all of the LEDs. A real application might want to do something different.
- 65** The line of code that actually performs the I/O operation.
- 72** More complete device support would probably do nothing more in this routine than register an `iocsh` command. The actual ASYN registration calls would then be called from this command when invoked by the `st.cmd` script. This would allow the port name to be set from the `st.cmd` script rather than being burned into the program.
- 80** The arguments to the `registerPort` method are:
- (a) The port name.
 - (b) The port attributes. This driver is not 'multi device' and does not block.
 - (c) The autoconnect flag. This driver wants to be automatically reconnected.
 - (d) The priority of the I/O thread (unused for this driver).
 - (e) The stack size of the I/O thread (unused for this driver).

9. Run `make` to compile the application.

10. Use the uCDIMM ColdFire 5282 setenv command to set environment variables as shown below. The exact values will differ as appropriate for your network numbers and NFS server:

```
B$ setenv IPADDR0 www.xxx.yyy.56
B$ setenv HOSTNAME ioccoldfire2
B$ setenv BOOTFILE ucdimm.boot
B$ setenv NAMESERVER www.xxx.yyy.167
B$ setenv NETMASK 255.255.252.0
B$ setenv SERVER www.xxx.yyy.167
B$ setenv NFSMOUNT nfserver:/export/homes:/home
B$ setenv CMDLINE ../FPGA_IOC_Example/EPICS/iocBoot/iocledDriver/st.cmd
B$ printenv
FACTORY=Arcturus Networks Inc.
REVISION=uC5282 Rev 1.0 4MB External Flash
SERIAL=X42B20ADC-0130C
CONSOLE=ttyS0
KERNEL=0:linux.bin
KERNEL_ARGS=root=/dev/rom0
HWADDR0=00:06:3B:00:53:0C
FW_VERSION=180001
_0=10000000:400000:RW
RAMIMAGE=yes
IPADDR0=www.xxx.yyy.56
CACHE=on
HOSTNAME=ioccoldfire2
BOOTFILE=ucdimm.boot
NAMESERVER=www.xxx.yyy.167
NETMASK=255.255.252.0
SERVER=www.xxx.yyy.167
NFSMOUNT=nfssrv:/export/homes:/home
CMDLINE=../FPGA_IOC_Example/EPICS/iocBoot/iocledDriver/st.cmd
```

11. Download and execute the application:

- Start the TFTP server on the ColdFire:

```
B$ tftp
uTFTP Console 1.0 is running ...
```

- Use the tftp program on your workstation to transfer the executable image to the ColdFire:

```
tftp> binary
tftp> connect www.xxx.yyy.56
tftp> put ledDriver.boot
```

- When the executable image has been transferred press the <ESC> key to the ColdFire to terminate the TFTP server. Use the goram command to start the IOC:

```
Downloading.....
B$ goram
Go from RAM!
Go from 0x40000
NTPSERVER environment variable missing -- using www.xxx.yyy.167

**** Initializing network ****
```

```

Startup after External reset.
fsl: Ethernet address: 00:06:3b:00:53:0c
***** Initializing NFS *****
This is RTEMS-RPCIOD Release $Name: $
($Id: tutorial.tex,v 1.17 2006/03/09 20:53:42 norume Exp $)

Till Straumann, Stanford/SLAC/SSRL 2002
See LICENSE file for licensing info
This is RTEMS-NFS $Name: $
($Id: tutorial.tex,v 1.17 2006/03/09 20:53:42 norume Exp $)

Till Straumann, Stanford/SLAC/SSRL 2002
See LICENSE file for licensing info
Trying to mount www.xxx.yyy.167:/export/homes on /home
***** Initializing NTP *****
***** Starting EPICS application *****
## Example RTEMS startup script
## You may have to change ledDriver to something else
## everywhere it appears in this file
#< envPaths
## Register all support components
dbLoadDatabase("../../dbd/ledDriver.dbd",0,0)
ledDriver_registerRecordDeviceDriver(pdbbase)
## Load record instances
dbLoadRecords("../..db/ledDriver.db","user=norume")
iocInit()
Starting iocInit
#####
### EPICS IOC CORE built on Jul 25 2005
### EPICS R3.14.7 $$Name: $$ $$Date: 2006/03/09 20:53:42 $$
#####
iocInit: All initialization complete
## Start any sequence programs
#seq snclDriver,"user=norume"
ioccoldfire2>

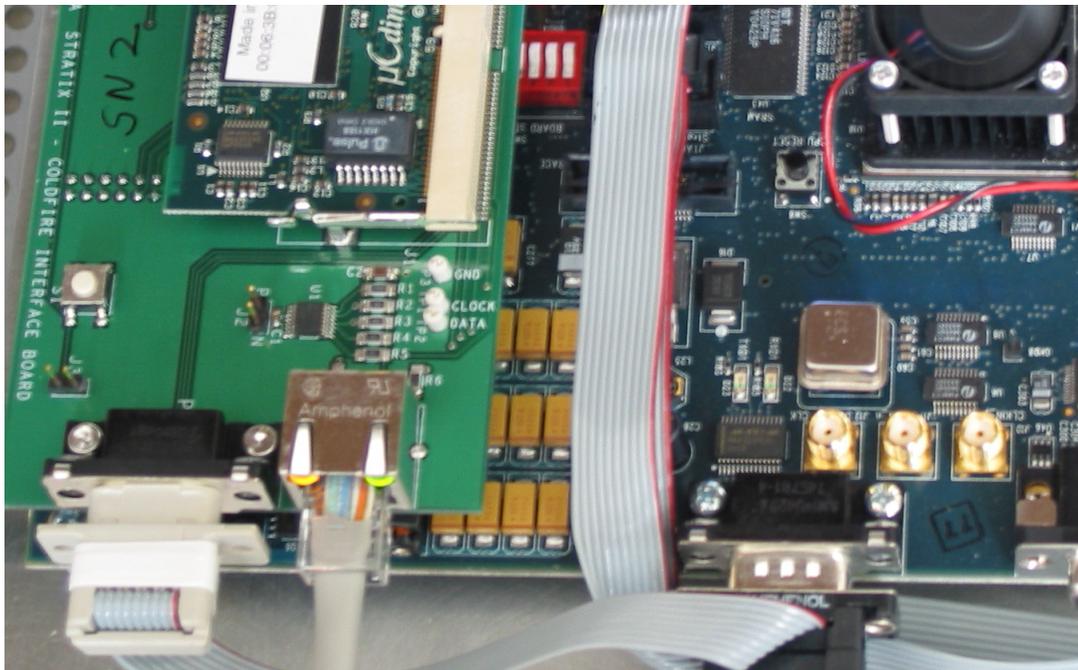
```

Chapter 4

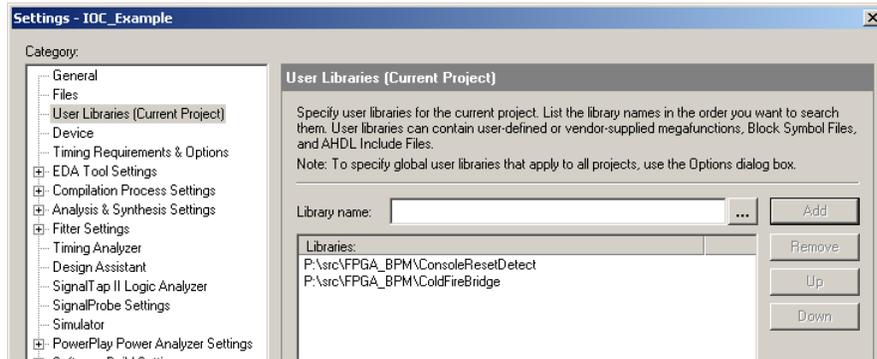
Remote Reset

All VME IOCs at the APS have a card which monitors the console received-data line and generates a system reset when a particular sequence (three consecutive control-X, control-Y or control-Z characters or any consecutive combinations of these characters) of characters is detected. It is quite easy to add this capability to the ColdFire/FPGA IOC.

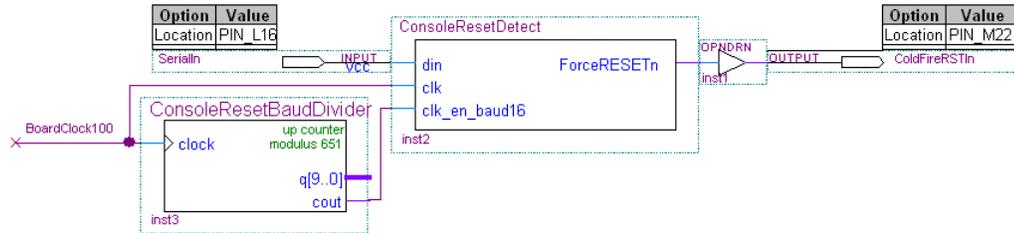
1. Make a special serial line cable which will connect the serial received-data and ground lines of the ColdFire console port to the 9-pin connector on the FPGA development kit. The following picture shows an example. The 9-pin connector plugged in to the FPGA development kit has only two pins (received-data and ground).



2. Add the path to the directory containing the Console Reset Detect component to the list of application user libraries:



3. Add the following components to the application top-level entity (`IOC_Example.bdf`). The `ConsoleResetBaudDivider` is a simple modulus-651 counter which sets the serial line speed at 9600 baud ($10000000 \div (9600 \times 16) = 651$). If you're using a different system clock or a different serial line speed you'll have to replace this component with your own counter.



4. Recompile the project and load it into the FPGA. You should now have the ability to remotely reset the ColdFire.

Chapter 5

Remote Reprogramming

This chapter describes the steps which add the ability to reprogram the flash memories on the the uCDIMM ColdFire 5282 module and the Altera FPGA development kits through EPICS channel access. This makes firmware updates of installed systems much easier.

5.1 uCDIMM ColdFire 5282 external flash

The uCDIMM ColdFire 5282 module provides 4 Mbytes of flash memory which can be used to store the RTEMS/EPICS executable and an in-memory file system providing the startup scripts and EPICS '.db' and '.dbd' files. Placing all these files in the flash memory allows the embeded IOC to be truly standalone, capable of starting and running even if there is no connection to the network. The flash memory can be programmed with the bootstrap 'program' command or, once the steps in this section have been applied, through EPICS channel access. Programming through channel access is convenient since it allows remote updates of multiple IOCs without the need to connect to each IOC console individually.

1. Edit configure/RELEASE to specify the location of the MCF5282 support:

```
MCF5282=/.../modules/instrument/mcf5282
```

2. Edit ledDriverApp/src/Makefile:

- Add mcf5282 support:

```
ledDriver_DBD += devFlashBurner.dbd
```

- Add the epicsMCF5282 library to ledDriver_LIBS (before the EPICS_BASE_IOC_LIBS and asyn lines):

```
ledDriver_LIBS += epicsMCF5282
ledDriver_LIBS += asyn
ledDriver_LIBS += $(EPICS_BASE_IOC_LIBS)
```

3. Edit ledDriverApp/Db/Makefile and add the line:

```
DB_INSTALLS += $(MCF5282)/db/xxdevFlashBurner.db
```

4. Edit iocBoot/iocledDriver/st.cmd:

- Add a line to configure the uCDIMM ColdFire 5282 module flash memory device support. The arguments to this command are the input device name, the base address of the flash memory, a boolean value where 0 means that the flash memory appears directly in the ColdFire address space, and the bit width of the flash memory:

```
devFlashBurnerConfigure("bootFlash",0x10000000,0,16)
```

- Add a line to load a flash burner record. The INP value must match the first argument to the configuration command:

```
dbLoadRecords("db/xxdevFlashBurner.db","P=led:,R=boot:,INP=bootFlash")
```

This will create a process variable named `led:boot:FlashBurner`.

5. Rebuild the application.

When the IOC starts up you should see something like the following when the flash memory burner is configured:

```
devFlashBurnerConfigure("bootFlash",0x10000000,0,16)
bootFlash: capacity 0x400000
bootFlash: erase block 0 size 0x2000 count 8 base 0
bootFlash: erase block 1 size 0x10000 count 63 base 0x10000
bootFlash: typical sector erase time 1.02 seconds (max 4.1 seconds)
```

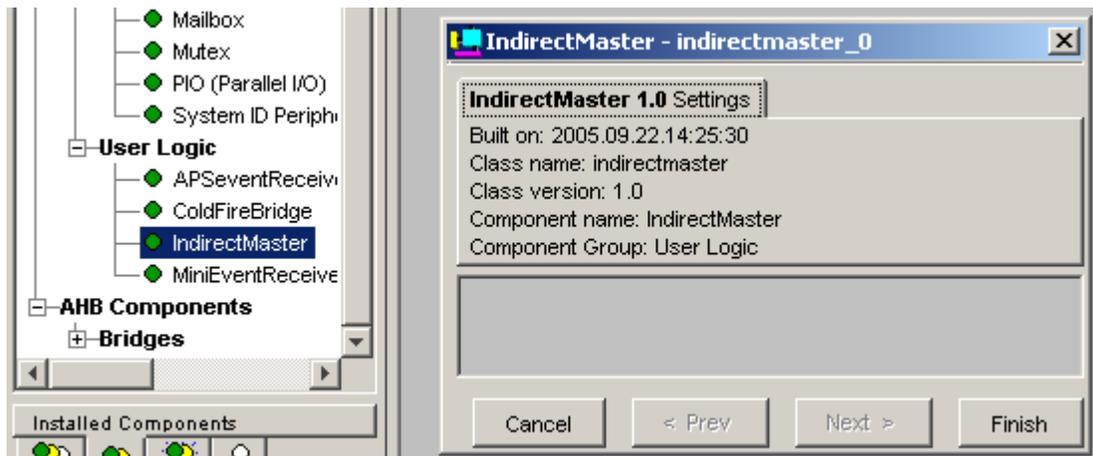
To burn a new image into the uCDIMM ColdFire 5282 flash memory module use the `flashBurner` command from the MCF5282 support area. The first argument to the command is the name of the flash burner PV in the ColdFire IOC. The second argument is the name of the file who's contents will be burned into the flash memory: a command like:

```
/.../modules/instrument/mcf5282/bin/hostArch/flashBurner \
    led:boot:FlashBurner bin/RTEMS-uc5282/ledDriver.boot
```

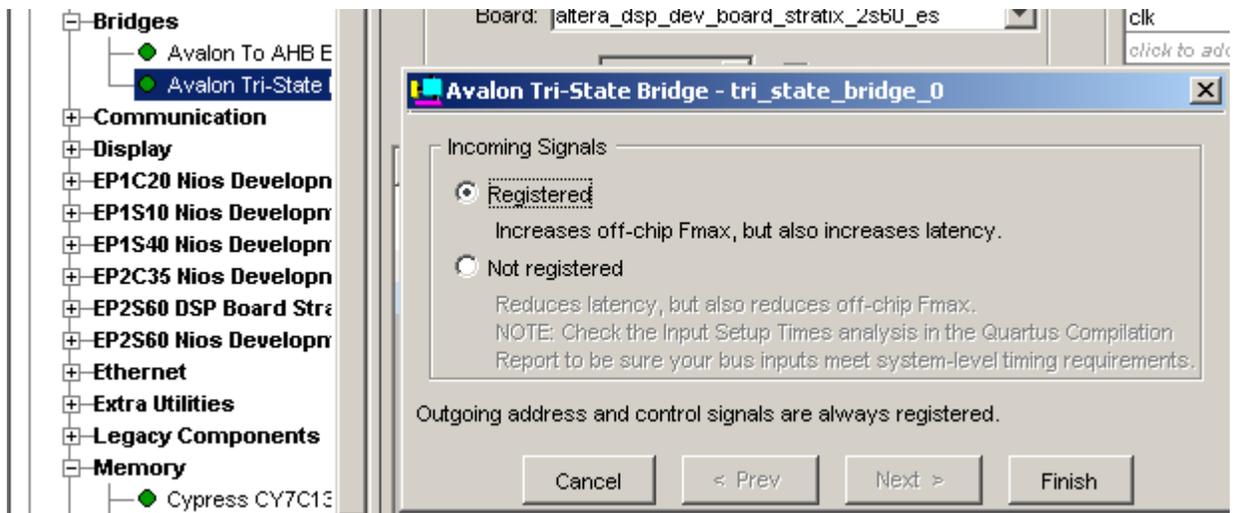
5.2 Development kit configuration flash

Adding the capability to remotely reprogram the FPGA configuration flash memory requires the following steps. The configuration flash memory can be connected directly to the ColdFire master but that eats up a large portion of the available address space. By using the IndirectMaster component the configuration flash can be connected to the ColdFire master without consuming 16 MB of address space.

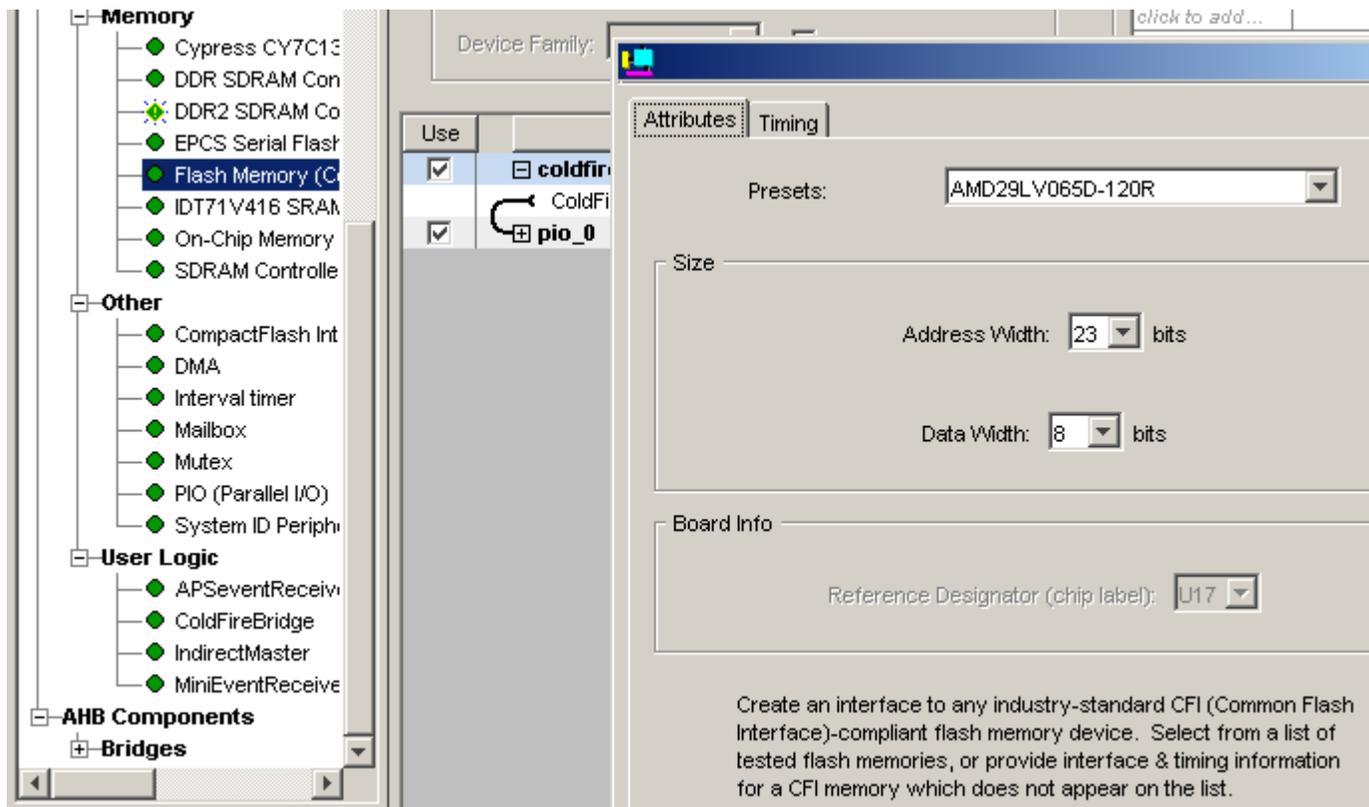
1. If you have not done so in a previous SOPC project, add the directory containing the Indirect Master SOPC component to the list of directories which the SOPC Builder will search.
2. Add an IndirectMaster component to the SOPC system:



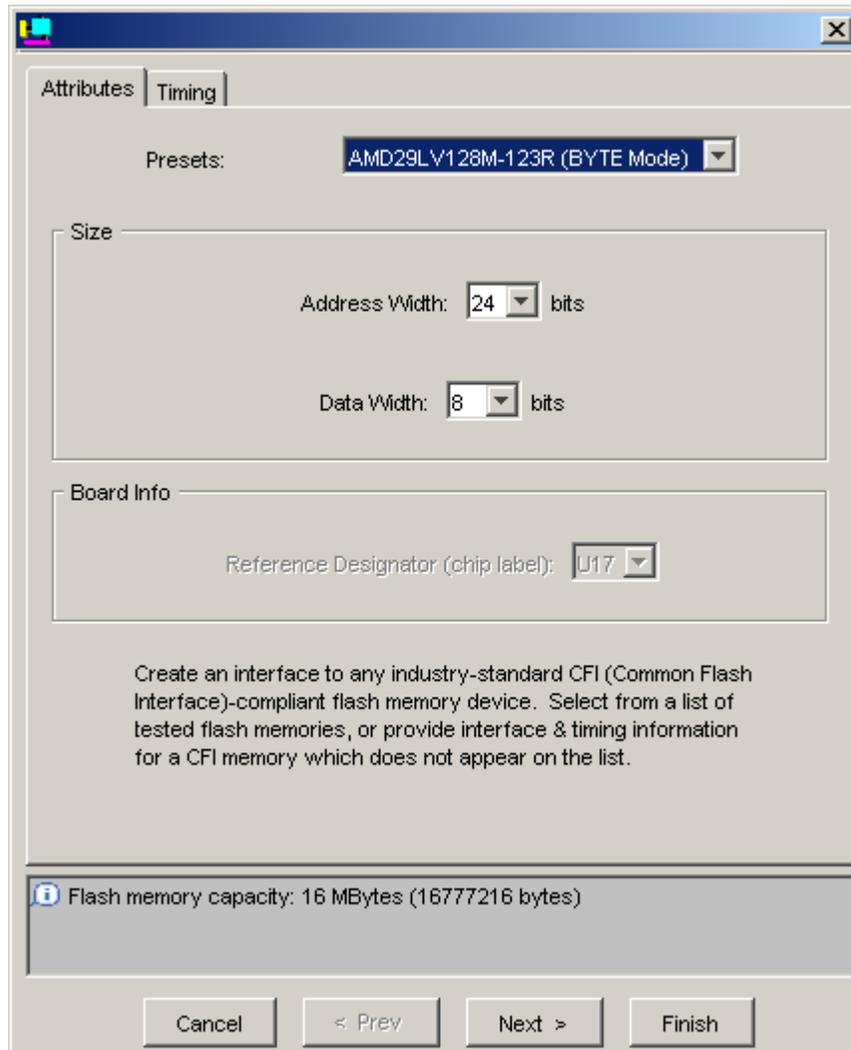
3. Add an Avalon Tri-State Bridge to the system



4. Add an Flash Memory (Common Flash Interface) component to the system



5. Select the Flash Memory configuration that matches the development kit hardware



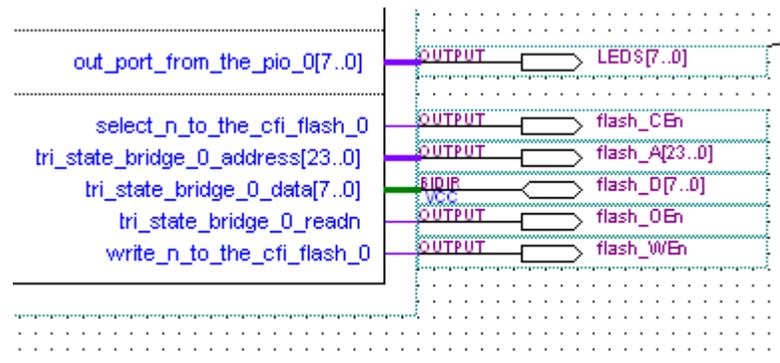
6. Connect the Avalon Tri-State bridge to the Indirect Master

Use	Module Name
<input checked="" type="checkbox"/>	coldfirebridge_0
	ColdFireMaster
<input checked="" type="checkbox"/>	pio_0
<input checked="" type="checkbox"/>	indirectmaster_0
	IndirectSlave
	IndirectMaster
<input checked="" type="checkbox"/>	tri_state_bridge_0
	avalon_slave
	tristate_master
<input checked="" type="checkbox"/>	cfi_flash_0

7. Set the IndirectMaster and Flash Memory addresses

Use	Module Name	Description	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>	<input type="checkbox"/> coldfirebridge_0	ColdFireBridge	clk			
	ColdFireMaster	Master port		IRQ 0	IRQ 63	
<input checked="" type="checkbox"/>	<input type="checkbox"/> pio_0	PIO (Parallel I/O)	clk	0x00000000	0x0000000F	
<input checked="" type="checkbox"/>	<input type="checkbox"/> indirectmaster_0	IndirectMaster	clk			
	IndirectSlave	Slave port		0x01FFFFC0	0x01FFFFCF	
	IndirectMaster	Master port				
<input checked="" type="checkbox"/>	<input type="checkbox"/> tri_state_bridge_0	Avalon Tri-State Bridge	clk			
	avalon_slave	Slave port				
	tristate_master	Master port				
<input checked="" type="checkbox"/>	<input type="checkbox"/> cfi_flash_0	Flash Memory (Common Flash Interface)		0x00000000	0x00FFFFFF	

8. Generate the SOPC system.
9. Update the IOC_Example block in the top-level design file. The signals to the flash memory should appear.
10. Add bidir and output pins to the IOC_Example flash memory signals.



11. Assign the pin numbers to the flash memory pins. The following table shows the flash memory pin assignments for the Stratix II DSP development kit.

Signal	Pin
flash_CEn	AA32
flash_OEn	AA31
flash_WEn	W32
flash_A[0]	AF30
flash_A[1]	AF29
flash_A[2]	AE30
flash_A[3]	AE29
flash_A[4]	AG32
flash_A[5]	AG31
flash_A[6]	AF32
flash_A[7]	AF31
flash_A[8]	AE32
flash_A[9]	AE31
flash_A[10]	AD32
flash_A[11]	AD31
flash_A[12]	AB28

flash_A[13]	AB27
flash_A[14]	AC32
flash_A[15]	AC31
flash_A[16]	AB30
flash_A[17]	AB29
flash_A[18]	Y29
flash_A[19]	Y28
flash_A[20]	AA30
flash_A[21]	AA29
flash_A[22]	AB32
flash_A[23]	AB31
flash_D[0]	AH30
flash_D[1]	AH29
flash_D[2]	AJ32
flash_D[3]	AJ31
flash_D[4]	AG30
flash_D[5]	AG29
flash_D[6]	AH32
flash_D[7]	AH31

12. Compile the system.

13. Edit `iocBoot/iocledDriver/st.cmd`:

- Add a line to configure the FPGA configuration module flash memory device support. The address is that of the IndirectMaster address assigned in the SOPC system plus the offset of the FPGA addresses in the ColdFire memory space. The width must match the development kit hardware flash memory:

```
devFlashBurnerConfigure("fpgaFlash", 0x31ffffc0, 1, 8)
```

- Add a line to load a flash burner record. The INP value must match the first argument to the configuration command:

```
dbLoadRecords("db/xxdevFlashBurner.db", "P=led:,R=fpga:,INP=fpgaFlash")
```

This will create a process variable named `led:fpga:FlashBurner`.

14. Rebuild the application.

When the IOC starts up you should see something like the following when the flash memory burner is configured:

```
devFlashBurnerConfigure("fpgaFlash", 0X31FFFFC0, 1, 8)
fpgaFlash: capacity 0x1000000
fpgaFlash: erase block 0 size 0x10000 count 256 base 0
fpgaFlash: typical sector erase time 1.02 seconds (max 16.4 seconds)
```

To burn a new image into the FPGA configuration flash memory the Quartus `.sof` file must be converted to a `.flash` file. Start the NIOS SDK shell and run the `sof2flash` program:

```
sof2flash --offset=0x00500000 --input=IOC_Example.sof --output=IOC_Example.flash
```

The offset value shown above is appropriate for the configuration flash memory on the Stratix II DSP development kit. The resulting `.flash` file is burned using the `flashBurner` command:

```
/.../modules/instrument/mcf5282/bin/hostArch/flashBurner \
    led:fpga:FlashBurner IOC_Example.flash
```


Appendix A

ledDeviceSupport.c

```
1  /*
2  * ASYN Int32 driver for simple FPGA application
3  */
4  #include <epicsStdio.h>
5  #include <epicsExport.h>
6  #include <cantProceed.h>
7
8  #include <asynDriver.h>
9  #include <asynInt32.h>
10 #include <devLib.h>
11
12 #include "SOPC.h" /* Symbolic link to SOPC-generated system header file */
13
14 #define AVALON_BASE 0x30000000 /* Base of Avalon space in ColdFire space */
15 #define OPTR ((epicsUInt8 *) (AVALON_BASE+PIO_0_BASE+1))
16
17 /*
18 * Driver private storage
19 */
20 typedef struct drvPvt {
21     asynInterface common;
22     asynInterface asynInt32;
23     asynInt32     asynInt32Methods;
24     volatile epicsUInt8 *optr;
25 } drvPvt;
26
27 /*
28 * asynCommon methods
29 */
30 static void
31 report(void *pvt, FILE *fp, int details)
32 {
33 }
34
35 static asynStatus
36 connect(void *pvt, asynUser *pasynUser)
```

```

37 {
38     drvPvt *pdrvPvt = (drvPvt *)pvt;
39     epicsUInt8 dummy = 0;
40
41     if (devWriteProbe(sizeof(dummy), pdrvPvt->optr, &dummy) != 0) {
42         asynPrint(pasynUser, ASYN_TRACE_ERROR, "ledDriver: memory probe failed\n");
43         return asynError;
44     }
45     pasynManager->exceptionConnect(pasynUser);
46     return asynSuccess;
47 }
48
49 static asynStatus
50 disconnect(void *pvt, asynUser *pasynUser)
51 {
52     pasynManager->exceptionDisconnect(pasynUser);
53     return asynSuccess;
54 }
55 static asynCommon common = { report, connect, disconnect };
56
57 /*
58  * asynInt32 methods
59  */
60 static asynStatus
61 int32Write(void *pvt, asynUser *pasynUser, epicsInt32 value)
62 {
63     drvPvt *pdrvPvt = (drvPvt *)pvt;
64
65     *pdrvPvt->optr = value;
66     return asynSuccess;
67 }
68
69 /*
70  * Register ourself with ASYN
71  */
72 static void ledDriverDeviceSupportRegistrar(void)
73 {
74     drvPvt *pdrvPvt;
75     asynStatus status;
76     const char *portName = "ledDriver";
77
78     pdrvPvt = callocMustSucceed(sizeof(drvPvt), 1, "ledDriver");
79
80     status = pasynManager->registerPort(portName, 0, 1, 0, 0);
81     if(status != asynSuccess) {
82         printf("ledDriver registerDriver failed\n");
83         return;
84     }
85
86     pdrvPvt->common.interfaceType = asynCommonType;
87     pdrvPvt->common.pinterface = (void *)&common;

```

```
88     pdrvPvt->common.drvPvt = pdrvPvt;
89     status = pasynManager->registerInterface(portName, &pdrvPvt->common);
90     if (status != asynSuccess) {
91         printf("ledDriver registerInterface failed\n");
92         return;
93     }
94
95     pdrvPvt->asynInt32Methods.write = int32Write;
96     pdrvPvt->asynInt32.interfaceType = asynInt32Type;
97     pdrvPvt->asynInt32.pinterface = &pdrvPvt->asynInt32Methods;
98     pdrvPvt->asynInt32.drvPvt = pdrvPvt;
99     status = pasynInt32Base->initialize(portName, &pdrvPvt->asynInt32);
100    if (status != asynSuccess) {
101        printf("ledDriver pasynInt32Base->initialize failed\n");
102        return;
103    }
104    pdrvPvt->optr = OPTR;
105 }
106 epicsExportRegistrar(ledDriverDeviceSupportRegistrar);
```


Appendix B

Alternate Pinouts

The information in this tutorial applies to several Altera development kits, including:

1. Stratix II DSP development kit(EP2S60), 100 MHz clock.
2. Stratix II NIOS development kit (EP2S60), "BoardClock100" input is a 50 MHz clock.
3. Cyclone II NIOS development kit (EP2C35), "BoardClock100" input is a 50 MHz clock, expansion prototype connectors require extensions to clear tall components on development kit.

The pin assignments for each of these kits are presented in the following table.

Signal	1	2	3
BoardClock100	A16	AF15	N2
ColdFireA[0]	R31	E8	E25
ColdFireA[1]	R30	J9	F24
ColdFireA[2]	P32	F8	F23
ColdFireA[3]	P31	A3	J21
ColdFireA[4]	M32	C4	J20
ColdFireA[5]	M31	C3	F25
ColdFireA[6]	N31	C5	F26
ColdFireA[7]	N30	K10	N18
ColdFireA[8]	L32	H9	P18
ColdFireA[9]	L31	G9	G23
ColdFireA[10]	M30	A5	G24
ColdFireA[11]	M29	B5	G25
ColdFireA[12]	N29	D6	G26
ColdFireA[13]	N28	A6	H23
ColdFireA[14]	L30	H10	H24
ColdFireA[15]	L29	K11	J23
ColdFireA[16]	K32	F10	J24
ColdFireA[17]	K31	A7	H25
ColdFireA[18]	K30	C7	H26
ColdFireA[19]	K29	D7	K18
ColdFireA[20]	J32	A8	K19
ColdFireA[21]	J31	G10	K23

ColdFireA[22]	H32	J11	K24
ColdFireA[23]	H31	F11	J25
ColdFireBS2n	U27	A24	AD19
ColdFireBS3n	U23	B24	AC19
ColdFireCLKOUT	T32	AC14	N26
ColdFireCS1n	V29	D18	AC20
ColdFireCS2n	V23	J18	AB20
ColdFireD[16]	AC27	H15	AE24 ¹
ColdFireD[17]	AC26	J15	T21
ColdFireD[18]	AD27	C16	V22
ColdFireD[19]	AD26	A17	AF23
ColdFireD[20]	Y23	C17	AE23
ColdFireD[21]	Y22	A18	AC22
ColdFireD[22]	Y25	F17	AB21
ColdFireD[23]	Y24	K16	AD23
ColdFireD[24]	AA27	G17	AD22
ColdFireD[25]	AA26	A19	AC21
ColdFireD[26]	Y27	C18	AD21
ColdFireD[27]	Y26	C19	AF22
ColdFireD[28]	W25	A20	AE22
ColdFireD[29]	W24	J17	V18
ColdFireD[30]	W27	A21	W19
ColdFireD[31]	W26	C20	U17
ColdFireIRQ1n	W28	A22	AF21
ColdFireOEn	U22	K17	AA17
ColdFireRSTIn	M22	H18	V17
ColdFireRSTOn	M23	J14	W17
ColdFireRpWn	U28	B22	AE20
ColdFireTAn	W29	C21	U18
ColdFireTEAn	L25	H17	AA18
ColdFireTSn	V28	C22	AF20
LEDS[0]	B4	AD26	AC10
LEDS[1]	D5	AD25	W11
LEDS[2]	E5	AC25	W12
LEDS[3]	A4	AC24	AE8
LEDS[4]	A5	AB24	AF8
LEDS[5]	D6	AB23	AE7
LEDS[6]	C6	AB26	AF7
LEDS[7]	A6	AB25	AA11
SerialIn	L16	H7	AB15

1. Pin AE24 on the Cyclone II kit is a dual-purpose pin and needs to be configured as Assignments→Device..., Device & Pin Options, Dual-Purpose Pins tab): Change nCEO from "Use as programming pin" to "Use as regular IO".